融合人工求解策略的数独回溯求解法 *

江顺亮，唐祎玲，徐少平，叶发茂

(南昌大学 计算机系，南昌 330031)

摘　要：数独有唯一解，回溯法可以保证获得正确结果。为了提高回溯法求解效率，向前搜索用最基础的人工策略进行求解，这样只需要两三个正确的候选数就可求解成功。基础人工策略求解的结果分为求解成功、求解失败和求解不确定三种情况，只有在求解不确定时才继续向前搜索，从而达到高效剪枝的目的；同时，在算法实施方面采用大量位运算。大量 9*9 数独的实验结果表明对于绝大部分数独，平均计算时间不超过 0.15 ms，对于那些极端困难的数独平均求解时间为 2 ms；另外，求解一个 16*16 数独的平均时间为 224 ms。通过实验还发现 17 个提示数的 9*9 数独数据集在各方面具有较好的分散性，建议作为标准测试用数据集。

关键词：数独；回溯法；人工策略；计算时间

中图分类号：TP301.5　　doi: 10.19734/j.issn.1001-3695.2018.08.0539

Sudoku backtracking algorithm with rule-base strategies

Jiang Shunliang, Tang Yiling, Xu Shaoping, Ye Famao

(*Dept. of Computer Science, Nanchang University, Nanchang* 330031, *China*)

Abstract: Sudoku has unique solution. Backtracking can guarantee the correct result. In order to improve the efficiency of backtracking method, Sudoku is solved by the most basic rule-base strategies during searching, only two or three correct candidates are needed to solve successfully. When the Sudoku is solved by the basic rule-base strategies, the returning result is one of 3 cases, i. e. the success, the failure, and the uncertainty. Only when the result is uncertain, the search is continued, as a result, the efficient pruning is achieved. Meanwhile, a number of bit operations are used in the algorithm implementation. A large number of 9x9 Sudoku experiments showed that the average calculation time is less than 0.15ms for most Sudoku and 2ms for those extremely difficult Sudoku. In addition, the average time to solve a 16x16 Sudoku is 224ms. The 9x9 Sudoku dataset with 17 cues is recommended as a standard test dataset due to its diversity.

Key words: Sudoku; backtracking; rule-base strategies; computational time

0　引言

数独不仅仅是风靡世界的益智填数游戏，而且已经渗透到多学科领域[1~9]。依据数独的数据分布特点，数独在图像数据隐藏[1]、秘密图像共享[2]和数字图像置乱技术[3]等方面有很好的应用。由于与拉丁方极其相似，数独在均匀设计有独特的优势[4]。近几年，有些学者开始将数独引入到人工神经网络领域，例如，构造可以从稀疏的已知数字恢复其他数字的神经网络，从而进行数独关联存储的技术[5]；结合数独游戏，澳大利亚学者研究了神经网络的技巧表达和获取技术[6]；在模拟式神经网络方面，构造可以学习数独规则的神经网络[7]。因此，对数独自身的研究具有有很好的学术价值。直到近几年，才确信 17 个已知数的数独是最少已知数的[8]。有学者研究了特殊数独的 NP 复杂性问题[9]，对数独与汉密尔顿回路之间的关系也进行了研究[10]。其他的研究集中在数独求解这方面[11~17]，因为它是数独生成和数独难度计算的基础[17]。有大量数独求解的进化类算法被开发出来[13,16]，文献[18]对这些算法进行了较为全面的比较；还有不少数学优化和搜索技术用于数独求解[12,14]，所有这两类方法的缺点是不能确保求解成功。其实，这些方法的计算效率还不如回溯法[11,15,17]，UCLA 的 Chi 比较了 3 种求解方法，其中的回溯法计算时间已经达到平均 7~9 ms[19]。

提高数独求解的回溯法效率主要在于提高剪枝的效率[11,15]，中国学者利用信息熵对候选数进行挑选[11]，略微超过了 UCLA 的 Chi 的计算效率[11,19]。肖华勇利用最基础的人工求解策略从最少候选数单元出发进行回溯也达到了不错的计算效率[15]。现有论文的回溯法有几个缺点，第一，没有很好地利用数独求解的特点进行算法的设计，数独有很多本身独有的一些特点，如何好好地利用现有的人工求解策略都没有涉及[11,17,19]，只是对搜索前进时选择什么样的单元[17,19]或选择什么样的候选数[11]进行探讨，文献[15]也只是简单地用基础人工求解策略对候选数进行筛选，并没有在搜索的过程中进行求解或者及时判断当前解不合格从而达到高效的剪枝；第二，数独是一种特有的数据结构，如何高效地组织数据从而提高算法效率都没有涉及，可以肯定现有回溯法的计算时间还有很大的减少空间；第三，没有标准的测试数据集，大部分是从网站收集一些数据[11,17,19]，或者只用几个测试用例的[15]。

本文旨在改进现有数独求解回溯法的计算效率。算法思想得益于数独方面非常活跃的 Andrew Stuart 发现的一个数

收稿日期：2018-08-04；**修回日期：**2018-09-25　**基金项目：**国家自然科学基金资助项目（61662044）

作者简介：江顺亮（1965-），男，江西丰城人，教授，博导，博士，主要研究方向为数字图像处理、计算机仿真、人工智能(jiangshunliang@ncu.edu.cn)；唐祎玲（1977-），女，浙江奉化人，讲师，博士研究生，主要研究方向为数字图像处理、图像质量评价、机器视觉；徐少平（1976-），男，江西九江人，教授，博导，博士，主要研究方向为图形图像处理、机器视觉、虚拟手术仿真等；叶发茂（1978-），男，江西金溪人，副教授，硕导，博士，主要研究方向为图像处理、人工智能.

独特点[20]，即绝大部分数独只要猜对了一两个数据就可以用最基础的人工策略求解成功。因此回溯法搜索前进时，用最基础的人工策略进行求解，可以预期这样的搜索深度会比较浅，从而达到高效剪枝的目的。另外，在算法实施方面尽可能采用位运算，通过这两方面的改进，在三种包含 50 000 个例子的数据集上的平均求解时间不到 0.1 ms，顺便给出了标准测试数据集的建议。

1　数据结构及基础算法的实现

为了叙述方便，先定义几个数独的习惯用词。标准数独共有 81 个单元格、9 行、9 列、9 宫（即 3x3 的九宫格，本文用宫替代），因此有 27 个块（9 行 9 列 9 宫），要用 1 到 9 填入每个空单元格中，使每个块中包含不重复的数字，同时填法是唯一的。未确定的单元格中还可以填入的数字叫“候选数”。数独开始时有不少于 17 个单元格确定了数字，此时的数独叫“初盘”，正确填完后叫“终盘”。

C 语言的位运算效率较高，不仅有非~、与&、或|，还有异或^和左右移位。为了大量地使用位运算，数独的数据结构设计如图 1 所示，单元格和宫的编号是行优先的。每个单元格有一个整数（bitLiveCell）来记录它的候选数，比如某单元格的该数是 001011001（二进制），表明“1”、“4”、“5”和“7”这四个数字是候选数。数组 bitLivRowDig 是用来记录某行某个候选数的分布，比如 bitLivRowDig[1][2]=101011000（二进制），表明第 2 行候选数“3”分布在第 4、第 5、第 7 和第 9 列。其他数据见图 1 中的注释，需要说明的是使用了 2 个布尔变量 solved 和 failed 表示当前解的状况，因为在搜索过程中数独可能求解成功也可能求解失败，还可能求解结果不确定，此时需要进一步向前搜索。

```
class Sudoku {
private:
    //单元格中的数字是否确定
    bool fixedCell[81];
    //单元格的候选数，一位对应一个数字
    int bitLiveCell[81];
    //一行中某个候选数的分布，一位对应一列
    int bitLivRowDig[9][9];
    //一列中某个候选数的分布，一位对应一行
    int bitLivColDig[9][9];
    //单元中候选数个数
    int numLiveCell[81];
    //一行中某个候选数的个数
    int numLivRowDig[9][9];
    //一列中某个候选数的个数
    int numLivColDig[9][9];
    //一宫中某个候选数的个数
    int numLivBoxDig[9][9];
public:
    int values[81]; //单元格中填入的数字
    iNt numFixed;  //已经确定的单元格数量
    bOol solved;   //数独是否成功求解
    bOol failed;   //数独是否求解失败
```

图 1　数独数据结构中的数据

Fig. 1　Variables in Sudoku data structure

为了提高运行效率，建立了 5 个全局索引数组。1)int boxid[81]，单元格所在宫的索引；2)int boxcell[9][9]，宫中单元格局部编号到数独单元格全局编号的索引；3)int dig2bit[9]，数字 k 到对应位数字 1<<(k-1)的索引；4)int bit2dig[1<<9]，对应位数字 1<<(k-1)到数字 *k* 的索引。如果计算的是 25x25 数独，这个数组就太大了，需要改为内置函数，但对于不大于 20x20 数独是没问题的；5)int popbit[1<<9]，二进制数中“1”的个数。

基础算法包括数独的基础人工求解策略[11,20]，具体包括：a) 唯一余数法 fixUniqueNumber ；b) 交差删除法一，pointingReduction，如果某个候选数只出现在某宫的一行或一列，可以删除该行或该列的其他位置的该候选数；c)交差删除法二，boxLineReduction，如果某个候选数只出现在某行或列的一个宫中，可以删除该宫的其他位置的该候选数；d)显性占位法，nakedMultiples ，如果一个块中的 k 个单元格只被 k 个候选数占据，可以删除该块中其他单元格的这些候选数；e)隐性占位法，hiddenMultiples ，如果一个块中的 k 个候选数只占据 k 个单元格，可以删除这些单元格中的其他候选数。

下面以显性占位法 nakedMultiples 为例，说明基础算法的编程实现情况。图 2 是显性占位法的核心代码，从一行中挑选的 ncons 个单元格放入数组 ijk，这个核心代码对这 ncons 个单元格进行判断，如果只被 ncons 个候选数占据，就删除其他单元格中的这些候选数。用位运算是比较适合这些判断的，用“或”运算对这 ncons 个候选数进行合并，同时合并它们的列，在后面进行删除操作时要排除这些列。合并后获得一个整数 livebit，从 livebit 中可以提取最右边的二进制“1”，它表示一个候选数（第 7 行），然后再从 livebit 中删去这个“1”（第 8 行），如果这样的操作进行了 ncons 次之后，livebit==0 意味着 ncons 个候选数占据 ncons 个单元格。当数独的当前解不合格时，有时会出现少于 ncons 个候选数占据 ncons 个单元格，这时可以置失败标志（第 12 行）。

```
1  livebit=colbit=0;
2  for(k=0; k<ncons; k++) {
3    //合并单元格的候选数
     livebit |= bitLiveCell[ ijk[k] ];
4    //合并单元格对应的列
     colbit |= dig2bit[ ijk[k]%9 ];
5  }
6  for(k=0; k<ncons; k++) {
7    //提取最右端的1
     delbit[k] = ( livebit & ~(livebit-1) );
8    //删除最右端的1
     Livebit = livebit & (livebit-1);
9  }
10 match=(livebit==0);   //删除后等于0吗
11 If(match) {           //匹配成功
12   //过少的候选数，解不合格
     if(delbit[ncons-1]==0)return failed=true;
13   for(c=0; c<9; c++) { //循环列，进行候选数删除
14     if(dig2bit[c]&colbit)continue; //此列被占据
15     cell=r*9+c;          //计算单元格编号
16     //这个单元格已经确定
       if(fixedCell[cell])continue;
17     for(k=0; k<ncons; k++) {   //循环删除候选数
18       //这个单元格有这个候选数
         if(bitLiveCell[cell]&delbit[k]) {
19         numChanges++;      //删除计数
20         //删除候选数
           remCellLive(cell,bit2dig[delbit[k]]);
21         //检验这个单元格是否可以确定
           testLiveCell(cell);
22       } //if bitLiveCell
23     } //k
24   } //c
25 } //if(match)
```

图 2　显性占位法核心代码及其注释

Fig. 2　Main codes and its comments of the naked candidate strategy

当确定一个单元格时，要删除对应块中的其他单元格对应的候选数。每次成功删除一个候选数都检验对应的单元格，因此确定一个单元格的函数是递归函数，有时确定一个单元格后整个数独求解成功。如果在删除候选数后，单元格的候选数为空，此时数独无解，通过设置失败标志 failed 及时剪枝。

2 融合人工求解策略的回溯求解法

一般来说，回溯法会进行大量的搜索，很多剪枝技术都是为了减少搜索的分枝从而提高计算效率的技术[11,15]。而本文的回溯法剪枝的方法是尝试对数独进行基础的人工求解策略求解，它的时间成本是非常高的。剪枝的时间成本和剪枝的效率对回溯法的最终时间效率是非常关键的，因此必须高效地实施人工求解策略，这也是上一节介绍数据结构和基础算法的原因。

本文回溯法剪枝的时间成本是较高的，因此只有剪枝的效率非常高才会有比较好的效果。Andrew Stuart 发现了数独的一个特点[20]，这个特点很好地支持了用基础人工求解策略进行剪枝的效率会非常高。这个数独特点就是：数独初盘只需要猜对非常少的单元格就可完全由基础的人工求解策略求解成功。Andrew Stuart 只是用这个特点对数独进行难度分级[20]。不需要猜就可由基础人工求解策略成功求解“0-级”数独，只需猜对 1 个单元格就可成功求解“1-级”数独，依次可以定义“2-级”和“3-级”数独。到目前为止，只发现一个“3-级”数独。“2-级”数独的比例也非常低，本文的实验收集了近 15,000 个数独，只有 37 个“2-级”数独。回溯法本身就是对解进行逐步地猜的过程，根据这个特点完全可以相信：搜索前进时用基础的人工求解策略求解只需少数几步搜索就可获得正确的解，后面的实验也证实了这一点。回溯法都是在尝试用基础的人工求解策略求解后使用，换言之，只有“1-级”及其更难级别的数独需要用到回溯法。

融合人工求解策略的回溯求解法与其他回溯求解法[11,15,19]的框架基本相同，不同的是每次向前搜索时会对数独进行求解，求解的方法是基础的人工求解策略。求解结果分三种:求解成功，直接返回；失败，换一个候选数搜索，如果没有候选数，则回溯；不确定是失败还是成功，此时要继续向前搜索。该方法的流程如图 3 所示。

为了更好地探讨本文方法的优劣和各种影响因素，下面介绍三种常规的回溯法。常规回溯法的算法描述如下：

输入：基础人工策略删除了部分候选数的数独

输出：终盘数独

a，如果处理完所有单元格，返回成功

b，选取一个候选数最少的单元格

  c，从该单元格顺序选取一个候选数

  d，如果没有候选数了，返回失败

e，用该候选数固定该单元格

f，更新其他单元的候选数，即删除相同块中的其他单元的该候选数

g，递归调用回溯法

  i，如果成功，返回成功

  j，如果失败，恢复数独到 e 之前的状态，转到 c

3 实验结果与分析

用 C++11 实现了本文算法及常规回溯法，操作系统是 Win10，开发平台是 Code::Block 16.01，电脑配置是 3.4 GHz CPU，内存 16 GB。

共有 11 组实验数据集，从互联网上搜集的不同人宣称的世界上最难数独共 20 例（数据集名“Extreme”），包括 Andrew Stuart 的 3 例[20]和 Arto Inkala 的 10 例（www.aisudoku.com），还有几例来自不同论坛，已经无法追踪了。从“数独之王”app 中手工收集了“Easy”、“Medium”、“Hard”及“Very Hard”4 个难度级别各 100 个数独。下载组合学家 Gordon Royle 收集的 17 个提示数的数独（staffhome.ecm.uwa.edu.au/~00013890/ sudokumin.php），共 49,151 个数独（数据集名“Su17”），从网站 sfsudoku.com 下载了 2 个各包含 50,000 数独的数据集（分别为“SS50k-1”和“SS50k-2”）。

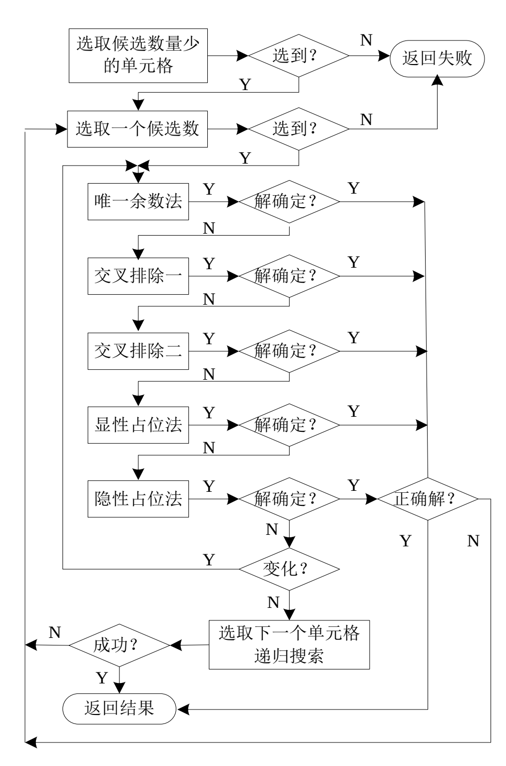


图 3 融合人工求解策略的回溯法流程图

Fig. 3. Flowchart of Backtracking Algorithm with Rule-Base Strategies

由于求解“0-级”数独时不需要调用回溯法，从数据集 Su17、SS50k-1 和 SS50k-2 中剔除“0-级”数独形成 3 个更小的数据集 Su17hard、SS50k-1h 和 SS50k-2h，所有这些数据集的详细信息如表 1 所示。需要说明的是，本文的基础人工策略比 Andrew Stuart 定义[20]的更基础，它不包括 3 数占位法和 4 数占位法；这样做是为了减少剪枝成本。

表 1 实验用数据集及其 Andrew Stuart 难度分级

Table 1 Experiment dataset and its andrew stuart difficulty level

数据集	数独数量	0-级数量	1-级数量	2-级数量	3-级数量
Extreme	20	0	1	18	1
Veryhard	100	47	53	1	0
Hard	100	76	24	0	0
Medium	100	97	3	0	0
Easy	100	100	0	0	0
Su17	49,151	41,588	7,547	16	0
Su17hard	7,563	0	7,547	16	0
SS50k-1	50,000	39,045	10,953	2	0
SS50k-2	50,000	38,900	11,100	0	0
SS50k-1h	10,955	0	10,953	2	0
SS50k-2h	11,100	0	11,100	0	0

为了更好地探讨方法的优劣和特点，共实现了 4 种回溯法。a）“回溯法 A”，即融合人工求解策略的回溯法，后面的图表中用“Fuse”注释;b）“回溯法 B”，上一节介绍的常规

回溯法，由于选择单元格和更新候选数，用“UpdSel”注释;c)“回溯法 C”，只顺序选取单元格，但更新候选数，用“Update”注释;d)“回溯法 D”,挑选单元格但不更新候选数，用“Select”注释。4 种回溯法、11 种数据集的求解数独的平均运行时间如表 2 所示。

表 2　不同回溯法的平均求解计算时间/ms

Table 2　Average running time of various backtracking methods /ms

数据集	A (Fuse)	B (UpdSel)	C (Update)	D (Select)
Extreme	2.000	4.250	56.275	7481.730
Veryhard	0.102	0.100	0.251	1.115
Hard	0.066	0.064	0.065	0.072
Medium	0.055	0.057	0.055	0.057
Easy	0.054	0.054	0.054	0.054
Su17	0.085	0.127	1.504	41.816
Su17hard	0.150	0.410	9.825	261.895
SS50k-1	0.071	0.067	0.084	0.251
SS50k-2	0.070	0.068	0.084	0.296
SS50k-1h	0.124	0.115	0.204	0.960
SS50k-2h	0.128	0.114	0.191	1.154

从表 2 可以看出回溯法的几个特点：a)对于难度级别不大的数独（easy/medium/hard），各种回溯法的计算效率几乎没有差别;b)对于难度级别非常大的数独（extreme/Su17hard），各种回溯法的计算效率有本质的区别，回溯法 A（Fuse）的计算时间只需回溯法 D（Select）的 0.05%,更新候选数相当于提前剪枝，这样与仅仅选择单元格相比提高了剪枝效率，但还是联合使用效果更好;c)回溯法 A（Fuse）与回溯法 B（UpdSel）相比，在求解高难度数独（Su17/Su17hard/Extreme）时的优势明显，求解其他数独只是略微多出不到 10%的计算时间。

3.1　与其他文献结果比较

稀疏优化求解法的论文[14]中列出了三种方法的平均计算时间，约束规划法 400 s，Sinkhorn 法 2～10 s，稀疏优化法 0.1s，与本文方法相差较大，没有可比性。

候选数优化法[15]与回溯法 A（Fuse）非常相似，主要有三点不同：a)两个方法的核心思想不同，回溯法 A（Fuse）利用数独的特点，减少搜索深度,候选数优化法是删除候选数从而减少分枝;b)回溯法 A（Fuse）会在向前搜索时反复用基础人工策略分析数独直至无法删除候选数，候选数优化法只是使用基础人工策略一次;c)从图 3 可以看出，回溯法 A（Fuse）尽早地判断数独解是成功还是失败，只有不确定时才向前搜索，候选数优化法没有做这样的处理。文献[15]选了 192 例数独，相当于本文的“Veryhard”数据集，计算时间在 10～22 ms（2.53 GHz CPU），另外，具体给出了 2 个例子，计算时间分别为 16ms 和 10ms。与之相对，回溯法 A（Fuse）计算“Veryhard”的平均时间为 0.102ms（3.4GHz CPU）,2 个例子的计算时间分别为 0.037ms 和 0.215ms。这样大的差别不可能纯粹由方法的不同造成，与编程实现还有很大关系，可惜文献[15]没有给出数据结构和基础算法。

回溯法 D（Select）与 Chi 的回溯法[19]相同。Chi 从“easy”“medium”和“hard”三个级别中删除“0-级”数独形成三个数据集，平均计算时间为 7～9 ms（3.4 GHz CPU）；这样的数据集相当于数据集 SS50k-1h 和 SS50k-2h，本文回溯法 A（Fuse）的计算时间为 0.1 ms（3.4GHz CPU），相同回溯法的计算时间为 1 ms。另外，Chi 从网站 warwick.ac.uk/fac/sci/moac/people/students/peter_cock/python/sudoku 下载了高难度数独的 Top95 数据集，本文四种回溯法的平均计算时间分别为：0.298 ms、0.651 ms、13.758 ms 和 232.747ms，其中回溯法 D（Select）与文献[19]中散点图的数据非常接近，这也说明了这两者的回溯法是相同的，而回溯法 A（Fuse）提速了将近 1000 倍。

用信息熵选择候选数的回溯法[11]平均计算时间为 3～6 ms（1.75GHz CPU），数据集是从网站 www.websudoku.com 收集的“easy”“medium”“hard”和“evil”四个难度级别数据。本文相对应的是“easy”“medium”、“hard”和“veryhard”四个数据集，回溯法 A（Fuse）的平均计算时间为 0.05～0.1 ms（3.4 GHz CPU）,考虑 CPU 的不同，回溯法 A（Fuse）的计算时间只需信息熵回溯法的 3%，即使回溯法 D（Select）的计算时间也不超过信息熵回溯法的 40%。由于本文四种回溯法都是在使用基础人工策略删去部分候选数后调用，因此性能比信息熵回溯法高是由两个因素造成：初始候选数的减少和编程实现的高效，而回溯法 A（Fuse）的剪枝性能进一步提升了性能。

3.2　回溯法的递归深度和递归次数

具有 17 个提示数的数独是提示数最少的[8]，但是根据研究与分析，最少提示数数独具有分布广泛性的特点[20]。针对 Su17hard 数据集，融合人工求解策略的回溯法的递归深度和递归次数见表 3，表 3 中的数据证实了只需少数几次猜对就可以用基础人工求解策略成功求解。由于每次都选择候选数最少的单元格进行搜索，不妨假设每次猜测都是从 2 个候选数中随机选一个，有 50%的概率选对，选对后能用基础人工求解策略成功求解的概率，从表 3 深度为 1 的数据（21.39%）推测，估计大概有 43%的概率。用这 2 个概率和平均递归次数（9.8）来估算，只需 2～3 次猜对了就可用基础人工求解策略求解成功，这与 Andrew Stuart 的结论[20]吻合。

其他常规回溯法都需要递归到最后一个单元格，所以递归深度一般都在 60 以上；远高于回溯法 A（Fuse）的递归深度。所以表 3 中没有列出回溯法 B（UpdSel）的递归深度，它的递归次数比回溯法 A（Fuse）高出了 3 个量级以上，这是常规回溯法计算性能不如融合人工求解策略的回溯法的主要原因。

表 3　回溯法的递归深度和递归次数统计（Su17hard）

Table 3 Recursive depth and recurrence statistics of backtracking methods

	回溯法 A(Fuse)				回溯法 B(UpdSel)	
	深度	百分率%	次数	百分率%	次数	百分率%
	1	21.39	1	21.39	2,000	17.31
	2	45.93	2	42.88	5,000	14.09
	3	15.11	3	14.11	10,000	12.84
	4	6.64	4	6.17	20,000	13.68
	5	3.17	5	3.15	40,000	13.34
	5+	7.75	5+	12.3	40,000+	28.73
平均值	5.4		9.8		60,087	
最大值	23		4,230		5,815,054	

3.3　算法复杂度分析

现有文献的数独求解法都是针对 9x9 数独的[11~20]，因此都没有进行算法复杂度分析。数独求解问题是 NP 问题[9]，数独求解法的算法复杂度比较高。另一方面，数独数据分布的随机性较大[3,4]，导致算法复杂度的分析比较困难。

为了分析算法复杂度，假设融合人工求解策略的回溯法的递归深度是 O(N)，其中 N 为数独矩阵的行数。合理分布

的 N 个单元格可以影响到所有单元格，因此这种假设有它的合理性，而且表 3 中的数据也表明回溯法 A(Fuse)的递归深度与 N 同量级。由于每次搜索时都会选取候选数最少的单元格进行，因此分枝数绝大部分是 2，少数是 3，极少数会出现 4，不妨设分枝数是 λ。人工求解策略复杂度最高的是 2 数显性占位法，它要对每行每列每宫的 2 单元格组合进行检查，匹配后要对块中其他单元格进行处理，因此它的复杂度是 O(N³)。综合这些分析，可以得到回溯法 A(Fuse)的算法复杂度为

$$O(N^3)\times\lambda^{O(N)}=O(N^3\alpha^N) \quad (1)$$

为了验证算法复杂度，对 44 种 16x16 的数独进行了求解，数独数据是从网站（magictour.free.fr/sudoku.htm）获得。求解数独的平均时间是 224 ms，取求解 9x9 数独的平均时间是 0.1ms，可以推出回溯法 A（Fuse）的时间复杂度是 $O(N^3 2.35^N)$，2.35 与分枝数非常吻合，表明算法复杂度分析是合理的。依据这个算法复杂度，求解一个 20x20 的数独需要 138s。作为比较，回溯法 B(UpdSel)需要递归到最后一个单元格，因此搜索深度是 $O(N^2)$。依据相同分析过程，可以获得它的算法复杂度是 $O(N\alpha^{N^2})$。回溯法 B(UpdSel)求解 16x16 数独的平均时间是 109s,几乎是回溯法 A（Fuse）的 500 倍。但是高阶数独的论文还不多，在其他文献中还没有发现这方面的数据。

3.4　Su17 数据集的广泛代表性

从表 1 可以看出，2-级难度数独是非常稀少，除非目的非常明确地搜寻，获得的概率是很低的，这也是不少人热衷于宣称发现世界最难数独[20]的原因。这些“世界最难的数独”也确实名副其实，“Extreme”数据集只有一个 1-级难度数独，而且这个数独在所有 1-级难度数独中是最难的，因为只有猜中唯一的一个单元格才能用基础人工求解策略求解成功。而在其他数据集中共有 149,591 个数独，只有 18 个 2-级难度数独，其中 Su17 就占 16 个。而且 Su17 的 1-级难度数独比例也不低，大略 15%。因此，Su17 数据集对不同难度的数独具有更好的多样性。

从求解时间上看，Su17 数据集也有更广泛的分布。在全部 149,611 个数独中，回溯法 A（Fuse）求解时间超过 10ms 的只有 4，其中“Extreme”1 个，“Su17”3 个；求解时间在[5,10]之间的共有 9 个，其中“Extreme”2 个，“Su17”7 个。求解时间超过 1ms 的共有 47 个，全部集中在“Extreme”和“Su17”2 个数据集。

从回溯法 A（Fuse）的递归深度看，“Su17hard”也比“SS50k-1h”和“SS50k-2h”分布广，分布结果见图 4。虽然回溯法 A（Fuse）的递归深度分布较广，但超过 10 的只有 157 个，占 2%，超过 16 只有 17 个，占 0.2%。与之相比，“SS50k-1h”的递归深度不超过 10，“SS50k-2h”的不超过 9。

“Su17”数据集不仅各个方面代表性比较好，而且容易获取，数独数量也足够，现共有 49,151 个（2018.7 数据）。因此，建议作为数独求解算法研究的标准数据集。

4　结束语

绝大部分数独只要猜对了一两个单元格的数字就可以用最基础的人工策略求解成功。这些最基础的人工求解策略包括：唯一余数法、交叉删除法、显性对数占位法和隐性对数占位法。因此回溯法搜索前进时用基础的人工策略进行求解，求解结果分为求解成功、求解失败和求解不确定 3 种情况，只有在求解不确定时才继续向前搜索，从而达到高效剪枝的目的。另外，在基础人工求解算法实施方面采用大量位运算。

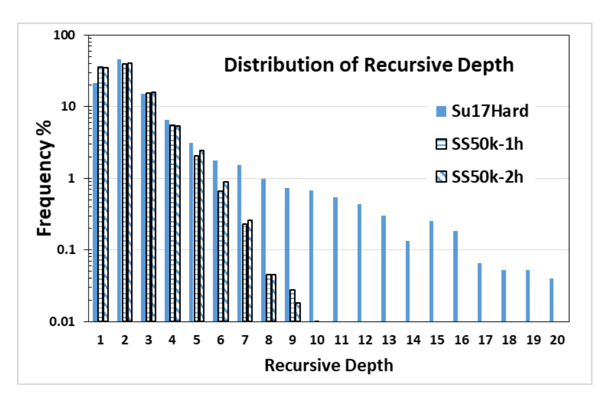


图 4　融合人工求解策略的回溯法的递归深度分布

Fig. 4　Recursive depth distribution of backtracking method with rule-base strategies

大量实验结果表明：对于绝大部分数独，平均计算时间不超过 0.15 ms，对于那些极端困难的数独平均求解时间为 2ms。通过实验还发现：提示数最少的数独数据集 Su17 具有各方面良好的多样性，建议作为数独求解算法的标准测试用数据集。

融合人工求解策略的回溯法在平均求解性能方面已经较好，如何降低最坏情况下的计算时间还有待于进一步研究。另外，挖掘出更多的 2-级难度和 3-级难度数独也许是一个值得研究的问题，也许可以促进数独在其他领域的应用。

参考文献：

[1] Nguyen T, Chang Chinchen. A reversible data hiding scheme based on the Sudoku technique [J]. Displays, 2015, 39: 109-116.

[2] 胡伟通，李明楚，郭成，等. 一种利用数独和细胞自动机的秘密图像共享方案 [J]. 小型微型计算机系统, 2015, 36 (06): 1271-1275. (Hu Weitong, Li Mingchu, Guo Cheng, *et al.* Secret image sharing scheme by Sudoku and cellular automata [J]. Journal of Chinese Computer Systems. 2015, 36 (06): 1271-1275.)

[3] Wu Yue, Zhou Yicong, Agaian Sos, *et al.* 2D Sudoku associated bijections for image scrambling [J]. Information Sciences, 2016, 327 (C): 91-109.

[4] Li Hongyi, Li Qisheng, Ou Zujun. Construction of Sudoku designs and Sudoku-based uniform designs [J]. Statistics & Probability Letters, 2014, 89 (89): 51-57.

[5] Wu Jiannming, Hsu Peihsun, Liou Chengyuan, *et al.* Sudoku associative memory [J]. Neural Networks, 2014, 57: 112-127.

[6] Leu G, Abbass H. Computational red teaming in a Sudoku solving context: neural network based skill representation and acquisition [J]. Proceedings in Adaptation, Learning and Optimization, 2016, 5: 319-332

[7] Barzegarjalali S, Parker A C. An analog neural network that learns Sudoku-like puzzle rules [C]//Proc of Future Technologies Conference. San Francisco, IEEE, 2016: 838-847.

[8] Mcguire G, Tugemann B, Civario G, *et al.* There Is No 16-Clue Sudoku: Solving the sudoku minimum number of clues problem via hitting set enumeration [J]. Experimental Mathematics, 2014, 23 (2): 190-217.

[9] Béjar R, Fernández C, Mateu C, *et al.* The Sudoku completion problem with rectangular hole pattern is NP-complete [J]. Discrete Mathematics, 2012, 312 (22): 3306-3315.

[10] Haythorpe M. Reducing the generalised Sudoku problem to the Hamiltonian cycle problem [J]. AKCE International Journal of Graphs & Combinatorics, 2016, 13 (3): 271-282.

[11] Zhai Gaoshou, Zhang Junhong. Solving Sudoku puzzles based on customized information entropy [J]. International Journal of Hybrid Information Technology, 2013, 6 (1): 77-92

[12] Musliu N, Winter F. A Hybrid approach for the Sudoku problem: using constraint programming in iterated local search [J]. IEEE Intelligent Systems, 2017, 32 (2): 52-62.

[13] G Singh, K Deep. A new membrane algorithm using the rules of particle swarm optimization incorporated within the framework of cell-like P-systems to solve Sudoku [J]. Applied Soft Computing, 2016, 45: 27-39

[14] 张煜东，王水花，霍元恺，等. 一种基于稀疏优化的数独求解新方法 [J]. 南京信息工程大学学报:自然科学版,2011, 3(1): 23-27. (Zhang Yudong, Wang Shuihua, Huo Yuankai, *et al.* A novel sudoku solving method based on sparse optimization [J]. Journal of Nanjing University of Information Science and Technology: Natural Science Edition, 2011, 3 (1): 23-27.)

[15] 程曦，肖华勇，吴林波. 数独求解的候选数优化算法设计 [J]. 科学技术与工程, 2011, 11(26): 6409-6412. (Cheng Xi, Xiao Huayong, Wu Linbo. The design of candidates optimization algorithm about sudoku puzzle [J]. Science Technology and Engineering, 2011, 11(26): 6409-6412.)

[16] Assad A, Deep K. Harmony search based memetic algorithms for solving sudoku [J]. International Journal of System Assurance Engineering & Management, 2017, 9(5): 1-14.

[17] 曲海平，岳峻，王飞. 数独问题的生成与求解算法的研究 [J]. 科技通报, 2017, 33(6): 14-17. (Qu Haiping, Yue Jun, Wang Fei. Study on generation and solution algorithm of sudoku problem [J]. Bulletin of Science and Technology, 2017, 33(6): 14-17.)

[18] Mishra D B, Mishra R, Das K N, *et al.* Solving Sudoku puzzles using evolutionary techniques—a systematic survey [M]//Soft Computing: Theories and Applications. Singapore, Springer, 2018:791-802.

[19] Chi E C, Lange K. Techniques for Solving Sudoku Puzzles [J]. arXiv preprint, 2012, arXiv: 1203. 2295.

[20] Andrew Stuart, A new metric for difficult Sudoku puzzles [N/OL]. 2017, http://www.sudokuwiki.org/A_New_Metric_for_Difficult_Sudoku_Puzzles.